Design Patterns

Introduction to Design Patterns

Eriq Muhammad Adams J.
Mail : eriq.adams@ub.ac.id | Blog : http://eriq.lecture.ub.ac.id

Origin & History

* 1970s an architect named Christoper Alexander carried out the first know work in the area of patterns.
* 1987 Kent Back & Ward Cunningham applied architectural pattern ideas for software design and development.
* 1994 GoF (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides) introduced 23 patterns.

Definition

* Design pattern is a documented best practice or core of a solution that has been applied successfully in multiple environments to solve a problem that recurs in a specific set of situations.

Design Patterns vs Frameworks

Design Patterns	*Frameworks*
Design patterns are recurring solutions to problems that arise during the life of a software application in a particular context.	A framework is a group of components that cooperate with each other to provide a reusable architecture for applications with a given domain.
The primary goal is to:	The primary goal is to:
• Help improve the quality of the software in terms of the software being reusable, maintainable, extensible, etc. • Reduce the development time	• Help improve the quality of the software in terms of the software being reusable, maintainable, extensible, etc. • Reduce development time
Patterns are logical in nature.	Frameworks are more physical in nature, as they exist in the form of some software.
Pattern descriptions are usually independent of programming language or implementation details.	Because frameworks exist in the form of some software, they are implementation-specific.
Patterns are more generic in nature and can be used in almost any kind of application.	Frameworks provide domain-specific functionality.
A design pattern does not exist in the form of a software component on its own. It needs to be implemented explicitly each time it is used.	Frameworks are not complete applications on their own. Complete applications can be built by either inheriting the components const directly.
Patterns provide a way to do "good" design and are used to help design frameworks.	Design patterns may be used in the design and implementation of a framework. In other words, frameworks typically embody several design patterns.

OO Basics

* Abstraction
* Encapsulation
* Polymorphism
* Inheritance

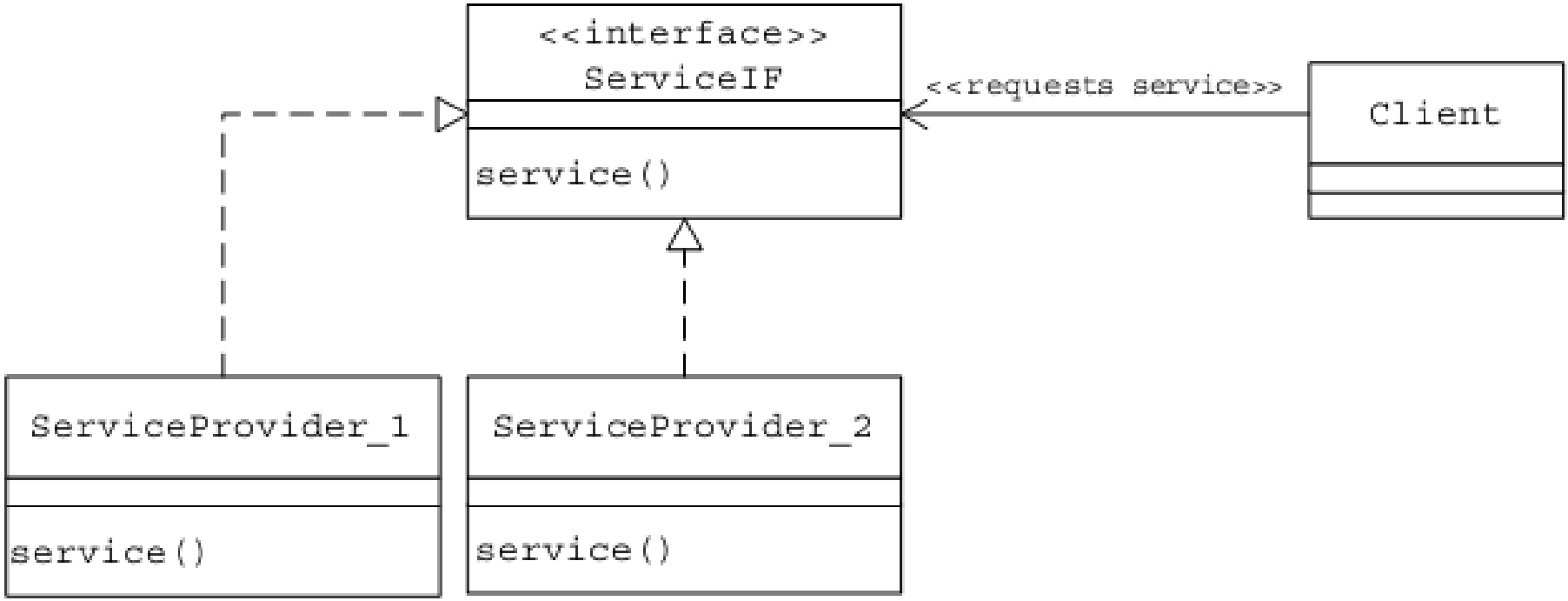
Design Principles

* Identify the aspects of your application that vary and separate them from what stays the same.
* Program to an interface, not an implementation.
* Favor composition over inheritance (has-a is better than is-a).

Basic Patterns

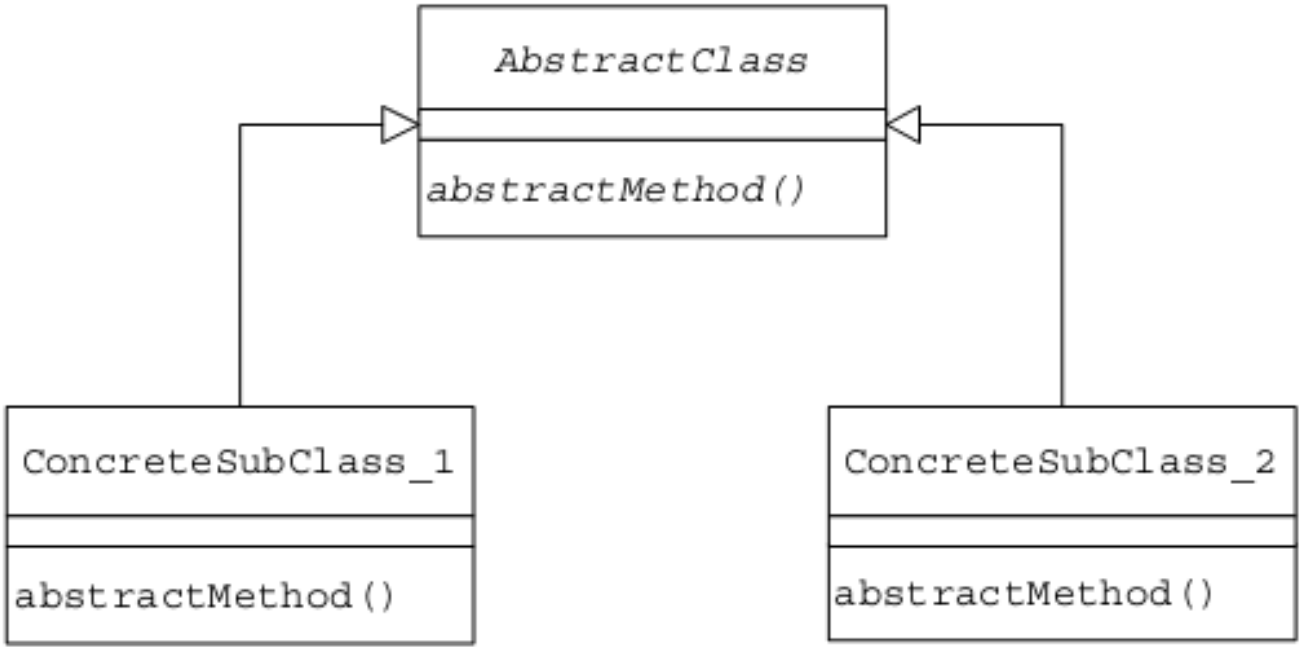
Pattern Name	*Description*
Interface	Can be used to design a set of service provider classes that offer the same service so that a client object can use different classes of service provider objects in a seamless manner without having to alter the client implementation.
Abstract Parent Class	Useful for designing a framework for the consistent implementation of the functionality common to a set of related classes.
Private Methods	Provide a way of designing a class behavior so that external objects are not permitted to access the behavior that is meant only for the internal use.
Accessor Methods	Provide a way of accessing an object's state using specific methods. This approach discourages different client objects from directly accessing the attributes of an object, resulting in a more maintainable class structure.
Constant Data Manager	Useful for designing an easy to maintain, centralized repository for the constant data in an application.
Immutable Object	Used to ensure that the state of an object cannot be changed. May be used to ensure that the concurrent access to a data object by several client objects does not result in race conditions.
Monitor	A way of designing an application object so that it does not produce unpredictable results when more than one thread tries to access the object at the same time in a multithreaded environment.

Interface Pattern



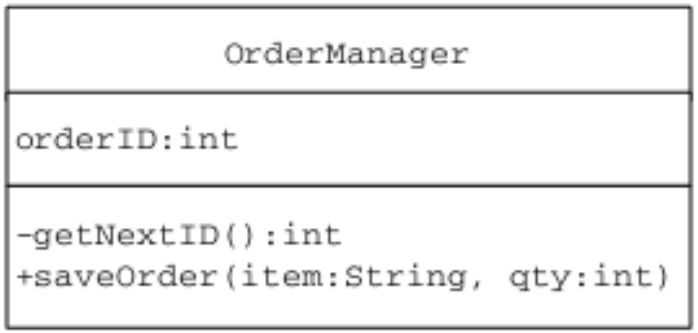
Common Interface with Different Service Providers as Implementers

Abstract Parent Class Pattern



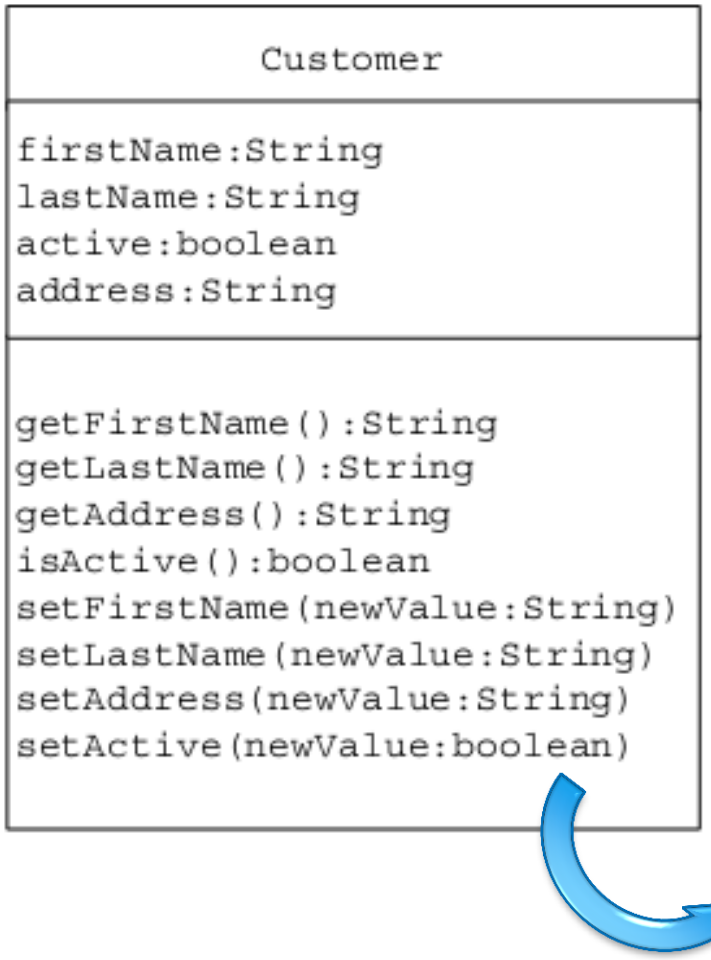
An Abstract Class with Two Concrete Subclasses

Private Methods Pattern




```
public class OrderManager {
  private int orderID = 0;
  //Meant to be used internally
  private int getNextID() {
    ++orderID;
    return orderID;
  }
  //public method to be used by client objects
  public void saveOrder(String item, int qty) {
     int ID = getNextID();
     System.out.println("Order ID=" + ID + ";
Item=" + item + "; Qty=" + qty + " is saved. ");
  }
}
```

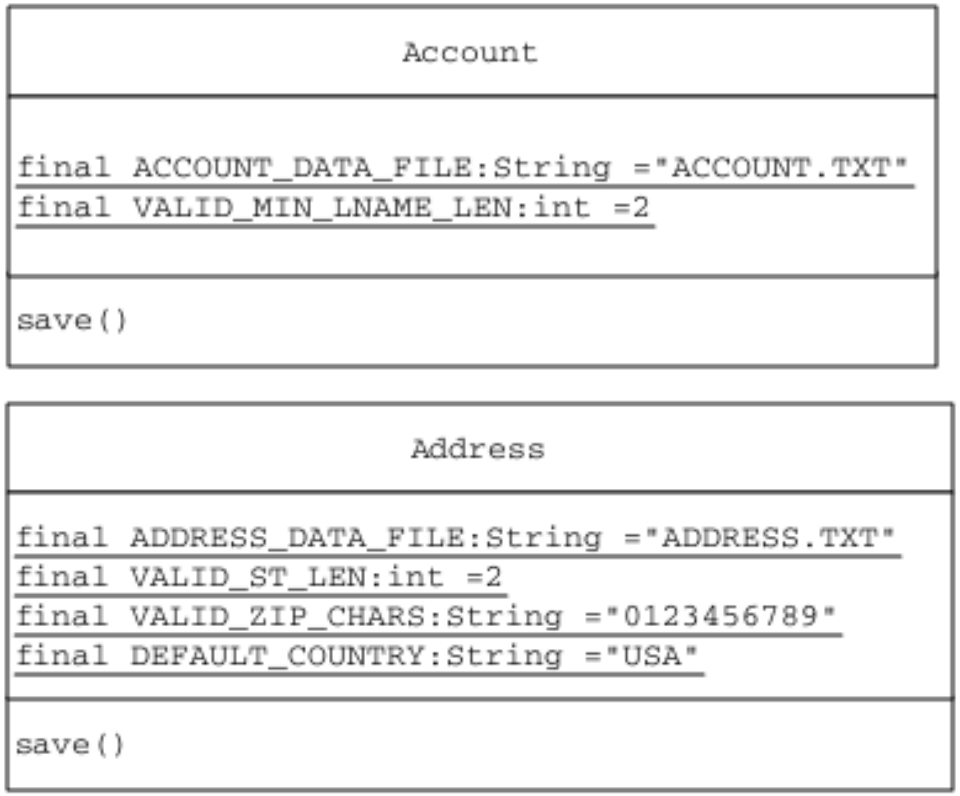
Accessor Method Pattern




```
public class Customer {
 private String firstName;
 private String lastName;
 private String address;
 private boolean active;
 public String getFirstName() { return firstName; }
 public String getLastName() { return lastName; }
 public String getAddress() { return address; }
 public boolean isActive() { return active; }
 public void setFirstName(String newValue) { firstName =
newValue;}
 public void setLastName(String newValue) { lastName =
newValue; }
 public void setAddress(String newValue) { address = newValue; }
 public void isActive(boolean newValue) { active = newValue; }
}
```

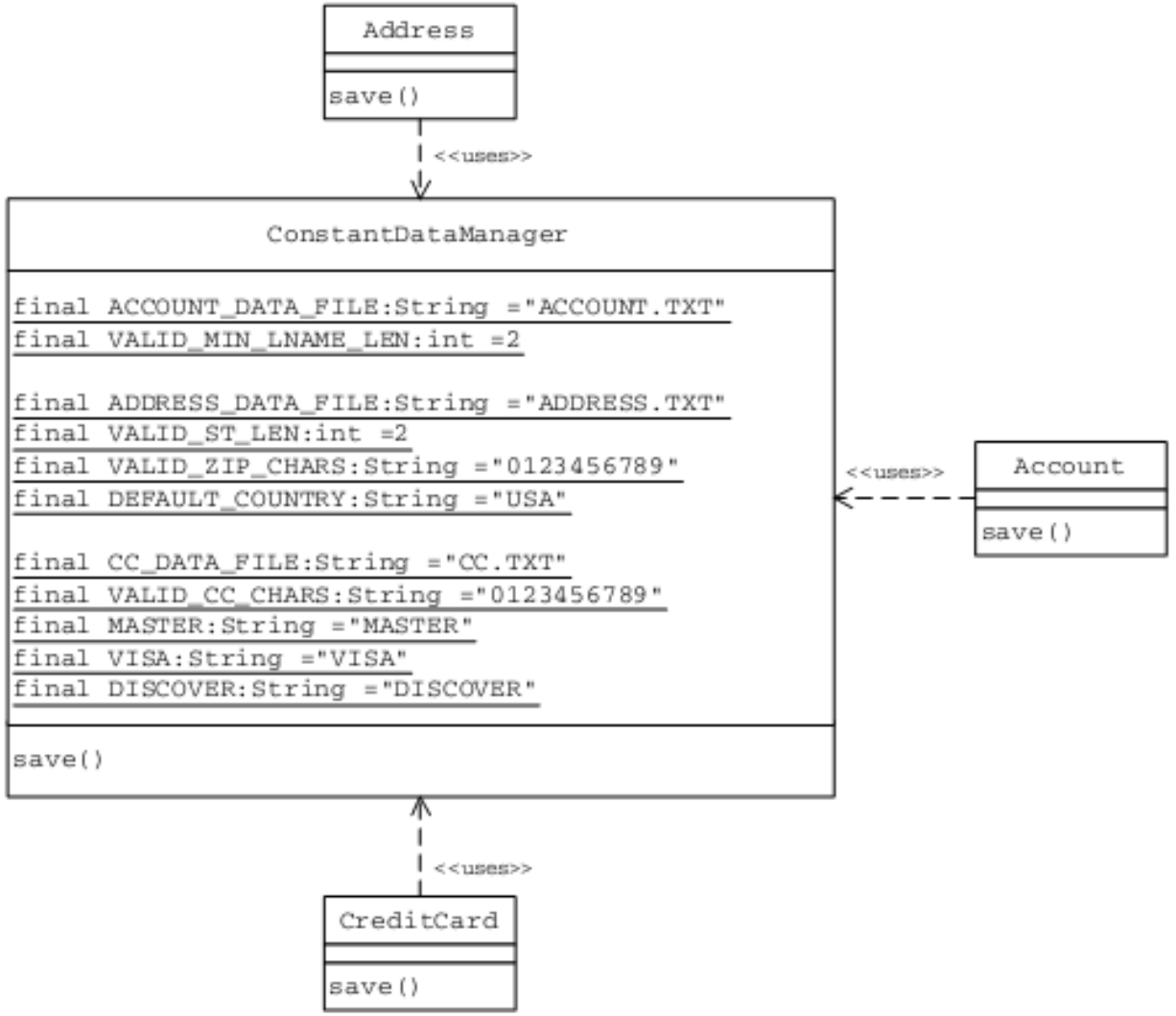
Constant Data Manager Pattern

Before Pattern

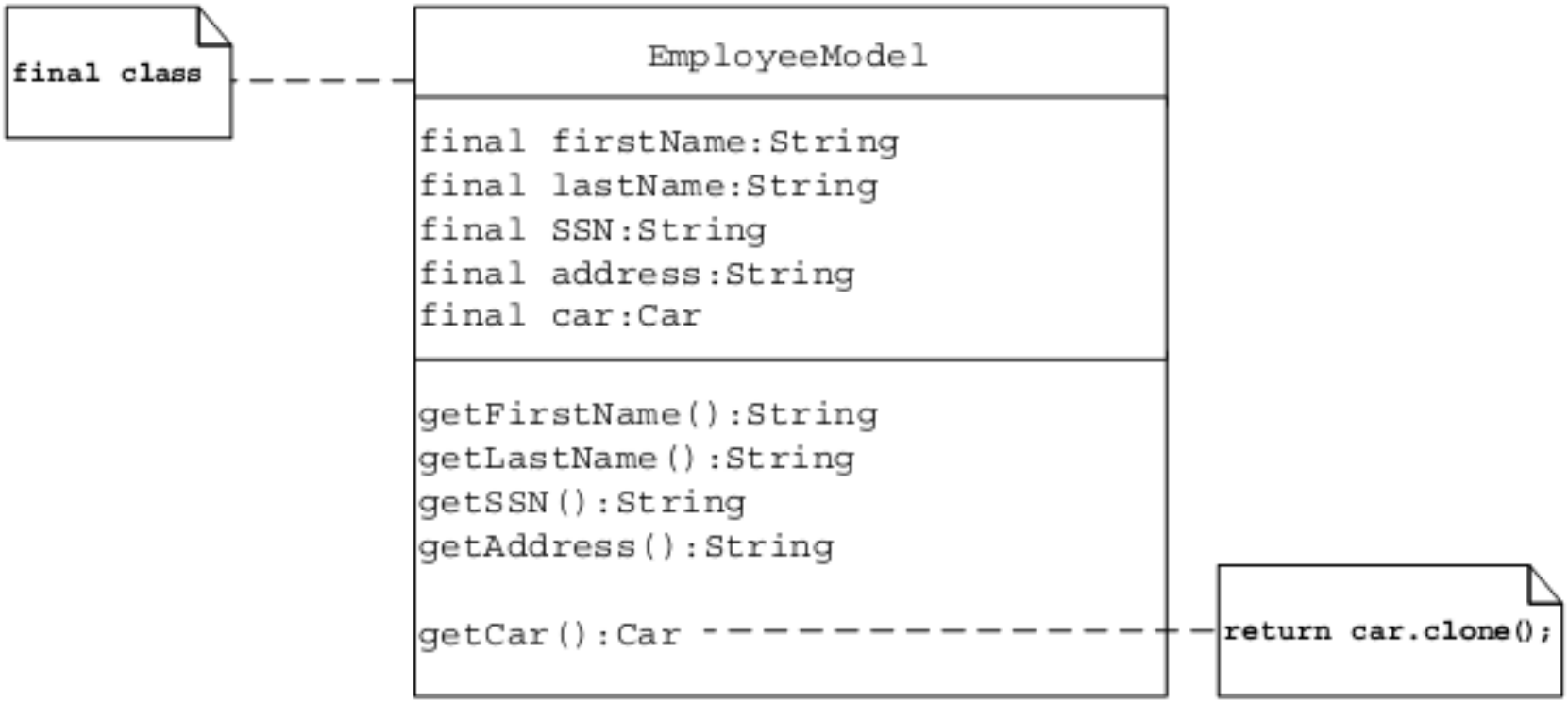


Constant Data Manager Pattern (cont.)

After



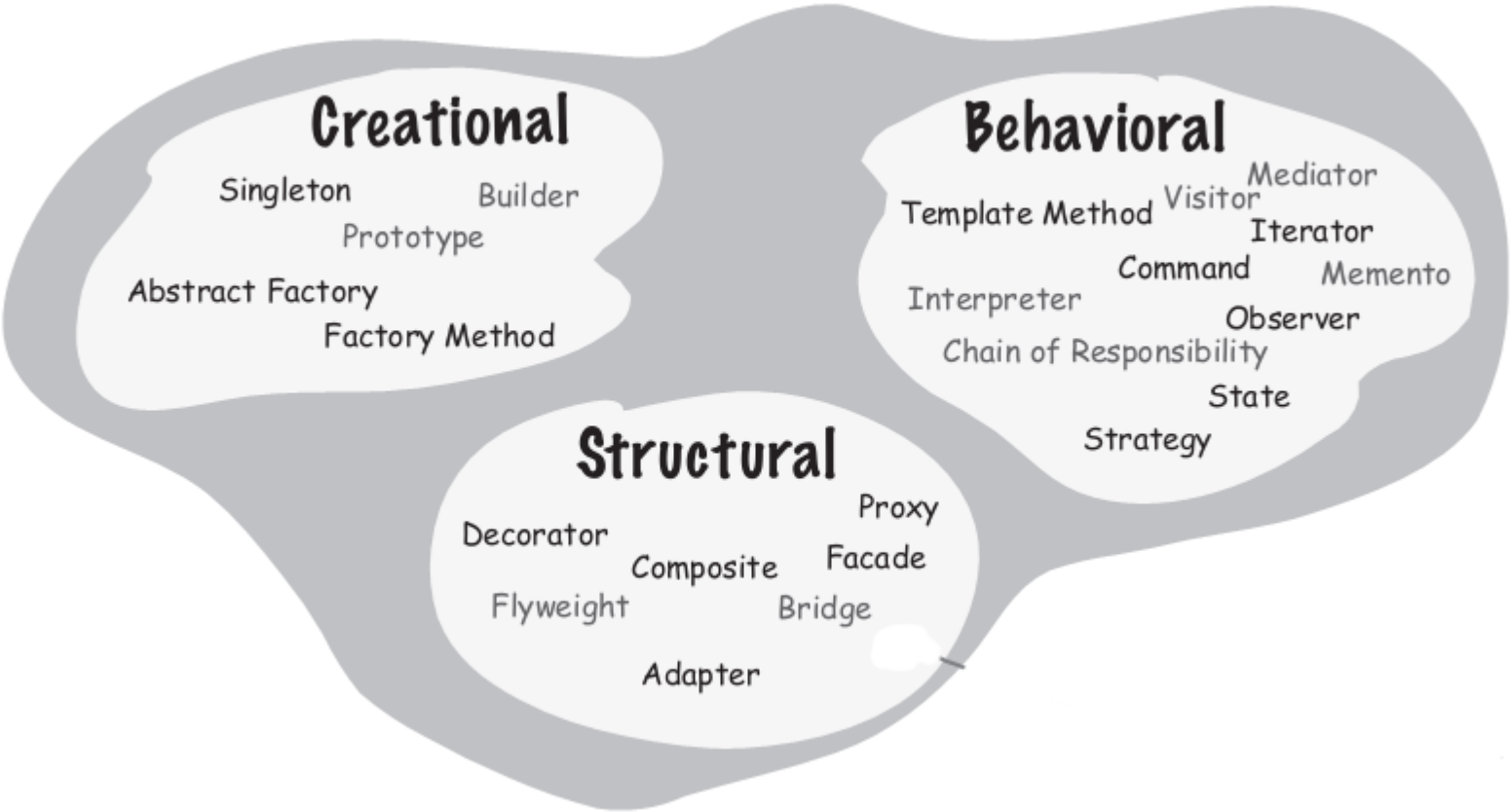
Immutable Object Pattern



Monitor Pattern

```
public class FileLogger {
   public synchronized void log(String msg) {
       DataOutputStream dos = null;
       try {
          dos = new DataOutputStream(
              new FileOutputStream("log.txt”,true));
          dos.writeBytes(msg);
          dos.close();
       } catch (FileNotFoundException ex) {
       } catch (IOException ex) { }
   }
}
```

GoF Patterns



References

* O'Reilly – Head First Design Pattern by Eric Freeman & Elisabeth Freeman (2004).
* CRC Press – Software Architecture Design Pattern in Java by Partha Kuchana (2004).